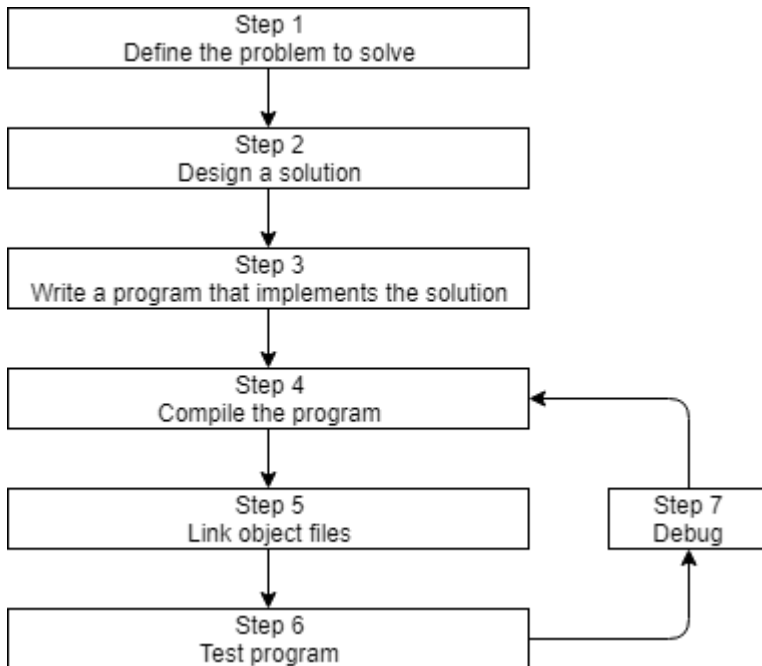




0.4 — Introduction to C++ development

👤 **ALEX¹** 🕒 **OCTOBER 25, 2023**

Before we can write and execute our first C++ program, we need to understand in more detail how C++ programs get developed. Here is a graphic outlining a simplistic approach:



Step 1: Define the problem that you would like to solve

This is the “what” step, where you figure out what problem you are intending to solve. Coming up with the initial idea for what you would like to program can be the easiest step, or the hardest. But conceptually, it is the simplest. All you need is an idea that can be well defined, and you’re ready for the next step.

Here are a few examples:

- “I want to write a program that will allow me to enter many numbers, then calculates the average.”
- “I want to write a program that generates a 2d maze and lets the user navigate through it. The user wins if they reach the end.”
- “I want to write a program that reads in a file of stock prices and predicts whether the stock will go up or down.”

Step 2: Determine how you are going to solve the problem

This is the “how” step, where you determine how you are going to solve the problem you came up with in step 1. It is also the step that is most neglected in software development. The crux of the issue is that there are many ways to solve a problem -- however, some of these solutions are good and some of them are bad. Too often, a programmer will get an idea, sit down, and immediately start coding a solution. This often generates a solution that falls into the bad category.

Typically, good solutions have the following characteristics:

- They are straightforward (not overly complicated or confusing).
- They are well documented (especially around any assumptions being made or limitations).
- They are built modularly, so parts can be reused or changed later without impacting other parts of the program.
- They are robust, and can recover or give useful error messages when something unexpected happens.

When you sit down and start coding right away, you’re typically thinking “I want to do <something>”, so you implement the solution that gets you there the fastest. This can lead to programs that are fragile, hard to change or extend later, or have lots of **bugs** (technical defects).

As an aside...

The term *bug* was first used by Thomas Edison back in the 1870s! However, the term was popularized in the 1940s when engineers found an actual moth stuck in the hardware of an early computer, causing a short circuit. Both the log book in which the error was reported and the moth are now part of the Smithsonian Museum of American History. It can be viewed [here](https://americanhistory.si.edu/collections/search/object/nmah_334663) (https://americanhistory.si.edu/collections/search/object/nmah_334663)³.

Studies have shown that only 20% of a programmer’s time is actually spent writing the initial program. The other 80% is spent on maintenance, which can consist of **debugging** (removing bugs), updates to cope with changes in the environment (e.g. to run on a new OS version), enhancements (minor changes to improve usability or capability), or internal improvements (to increase reliability or maintainability).

As an aside...

This is a great illustration of the [Pareto principle](https://en.wikipedia.org/wiki/Pareto_principle) (https://en.wikipedia.org/wiki/Pareto_principle)⁴.

Consequently, it’s worth your time to spend a little extra time up front (before you start coding) thinking about the best way to tackle a problem, what assumptions you are making, and how you might plan for the future, in order to save yourself a lot of time and trouble down the road.

We’ll talk more about how to effectively design solutions to problems in a future lesson.

Step 3: Write the program

In order to write the program, we need two things: First, we need knowledge of a programming language -- that’s what these tutorials are for! Second, we need a text editor to write and save

our written programs. The programs we write using C++ instructions are called **source code** (often shortened to just **code**). It's possible to write a program using any text editor you want, even something as simple as Windows' notepad or Unix's vi or pico.

A program typed into a basic text editor would look something like this:

```
#include <iostream>

int main()
{
    std::cout << "Here is some text.";
    return 0;
}
```

However, we strongly urge you to use an editor that is designed for programming (called a **code editor**). Don't worry if you don't have one yet. We'll cover how to install a code editor shortly.

A typical editor designed for coding has a few features that make programming much easier, including:

1. Line numbering. Line numbering is useful when the compiler gives us an error, as a typical compiler error will state: *some error code/message, line 64*. Without an editor that shows line numbers, finding line 64 can be a real hassle.
2. Syntax highlighting and coloring. Syntax highlighting and coloring changes the color of various parts of your program to make it easier to identify the different components of your program.
3. An unambiguous, fixed-width font (often called a "monospace font"). Non-programming fonts often make it hard to distinguish between the number 0 and the letter O, or between the number 1, the letter l (lower case L), and the letter I (upper case i). A good programming font will ensure these symbols are visually differentiated in order to ensure one isn't accidentally used in place of the other. All code editors should have this enabled by default, but a standard text editor might not. Using a fixed-width font (where all symbols have the same width) makes it easier to properly format and align your code.

Here's an example of a C++ program with line numbering, syntax highlighting, and a fixed-width font:

```
1 | #include <iostream>
2 |
3 | int main()
4 | {
5 |     std::cout << "Here is some text.";
6 |     return 0;
7 | }
```

Note how much easier this is to understand than the non-highlighted version. The source code we show in this tutorial will have both line numbering and syntax highlighting to make that code easier to follow.

Tip

[Coding Font](https://www.codingfont.com/) (<https://www.codingfont.com/>)⁵ has a neat tool to allow you to compare different coding fonts to see which ones you like best.

For advanced readers

Because source code is written using ASCII characters, programming languages use a certain amount of ASCII art to represent mathematical concepts. For example, `≠` is not part of the ASCII character set, so programming languages typically use `≠` to represent mathematical inequality instead.

Some programming fonts, such as [Fira Code](https://github.com/tonsky/FiraCode) (<https://github.com/tonsky/FiraCode>)⁶, use ligatures to combine such “art” back into a single character. For example, instead of displaying `≠`, Fira Code will display `≠` (using the same width as the two-character version). Some people find this easier to read, others prefer sticking with a more literal interpretation of the underlying characters.

The programs you write will typically be named `something.cpp`, where `something` is replaced with the name of your choosing for the program (e.g. calculator, hi-lo, etc...). The `.cpp` extension tells the compiler (and you) that this is a C++ source code file that contains C++ instructions. Note that some people use the extension `.cc` (or `.cxx`) instead of `.cpp`, but we recommend you use `.cpp`.

Best practice

Name your code files `something.cpp`, where `something` is a name of your choosing, and `.cpp` is the extension that indicates the file is a C++ source file.

Also note that many complex C++ programs have multiple `.cpp` files. Although most of the programs you will be creating initially will only have a single `.cpp` file, it is possible to write single programs that have tens or hundreds of `.cpp` files.

Once we’ve written our program, the next steps are to convert the source code into something that we can run, and then see whether it works! We’ll discuss those steps (4-7) in the next lesson.



Next lesson

0.5 [Introduction to the compiler, linker, and libraries](#)

7



Back to table of contents

8



Previous lesson

0.3 [Introduction to C/C++](#)


9

10



B **U** **URL** **INLINE CODE** **C++ CODE BLOCK** **HELP!**

Leave a comment...

 Name*

@ Email* | ?

Notify me about replies:



POST COMMENT

🔧 Find a mistake? Leave a comment above!?

👤 Avatars from <https://gravatar.com/>¹² are connected to your provided email address.

297 COMMENTS

Newest ▼

**Qwerty**

🕒 February 16, 2024 11:00 am

What are the differences between .cpp, .cc, and .cxx and why do you recommend .cpp (also why do people use the other two?)

👍 3

➡ Reply

**Alex** Author🗨 Reply to [Qwerty](#)¹³ 🕒 February 17, 2024 6:04 pm

There is no difference, they are just different conventions that people have various preferences for.

We recommend .cpp because it is the most common. People use the other two because they can.

👍 12

➡ Reply

**saeed**

🕒 February 4, 2024 11:16 pm

Thanks for your good site

🔗 *Last edited 2 months ago by saeed*

👍 7

➡ Reply

**Piyush Yadav**

🕒 February 2, 2024 4:49 am

My preference

Code editor : VS CodeFont family : Fira CodeFont Size : 16Theme : Github DarkIcon Pack : Material Icon Theme

👍 2

➡ Reply

**Neptune**🗨 Reply to [Piyush Yadav](#)¹⁴ 🕒 March 3, 2024 1:22 pm

thanks for that, I'll give it a try !

👍 0

➡ Reply

**S.E.**

🕒 January 23, 2024 5:39 am

I landed on Fira Mono with the side that waws suggested, though I think I will also try Fira Code. I hope it doesn't lead to bad habits.

👍 0

➡ Reply

**IceFloe**

🕒 January 18, 2024 5:15 pm

What should I do if the characters are not in ASCII, for example Cyrillic and Arabic characters, which I can use to, for example, translate the game from other languages, do I need to additionally install extensions and other libraries?

👍 1

➡ Reply

**Kamil G.**🗨 Reply to [IceFloe](#)¹⁵ 🕒 January 24, 2024 5:31 am

I don't know how it is for Cirillic and Arabic characters. But when I want to output Polish characters in my program, at the begining inside my `main()` function I write this line of code:

```
setlocale(LC_ALL, "polish");
```

and it works, so it is enough for me.

📄 Last edited 2 months ago by Kamil G.

👍 0

➡ Reply

**Alex** Author🗨 Reply to [IceFloe](#)¹⁵ 🕒 January 19, 2024 9:28 am

This is a bit of a complex topic, and I'm not currently knowledgeable enough to advise on best practices here.

Many third party libraries (e.g. QT) come with localization/internationalization support. I'd guess using one of those is probably your best bet.

👍 1

➡ Reply

**sopheak**

🕒 January 4, 2024 6:52 pm

thank you for your tutorial. I like study c++ code.



1

➡ Reply



Fahad Hossain Babor

🕒 December 27, 2023 10:18 pm

Hi Alex! Thanks for your efforts. I really appreciate your work. But the videos you putted seems little bit annoying ...Can you please reconsider that?



1

➡ Reply



Oliver

🕒 December 17, 2023 7:18 pm

I've decided to do 20 pushups between each chapter and will document them here with the number of pushups as well as my thoughts on that chapter if i feel like it.

// number of pushups on chapter 4

int pushups = 20



21

➡ Reply



abnv

🕒 December 16, 2023 10:26 pm

hello. your tutorials are very good.

I had this question in mind, there are 256 characters in ascii so any of those when used while writing code will they work.

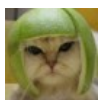
also since != is used as not equal to then how the computer knows its not equal to.

thank you.



0

➡ Reply



Alex Author

🗨 Reply to [abnv](#)¹⁶ 🕒 December 19, 2023 7:57 pm

1. There are only 128 characters in ASCII. The other 128 are "extended ASCII". I think you can use extended ASCII in your code (I tried naming a variable `f` and it worked), but use of such characters for outputting may not display correctly.
2. When the compiler sees `≠` in sequence, it has rules that enable it to identify that sequence as the "not equals" operator. See

<https://www.techtarget.com/searchapparchitecture/definition/parser> for some more info on how this works.



1



Reply

**AbeerOrTwo**

🕒 December 13, 2023 6:35 pm

Legggggooooo



2



Reply

**rax**

🕒 October 29, 2023 5:44 pm

What sort of syntax highlighting is used in the code examples? I would like to replicate that in my ide.

Last edited 5 months ago by rax



0



Reply

**Alex** Author Reply to [rax](#) ¹⁷ 🕒 October 30, 2023 12:37 pm

We're currently using prism.js to do syntax highlighting, modified to look like Visual Studio.

If you like the style, see if someone on the internet has posted Visual Studio visual settings for your IDE.



1



Reply

**Georgi**

🕒 October 20, 2023 3:34 am

Great tutorial, Alex!

I think that in the section -> Step 3: Write the program it's a good idea to give an example without ide -> no highlighting, no line numbers, etc. & in contrast to this -> show the current example (ide like).

Thanks for these wonderful resources!



0



Reply

**Alex** AuthorReply to [Georgi](#)¹⁸ October 21, 2023 1:23 pm

Done. Thanks for the suggestion.

👍 1

Reply

**Krishnakumar**

October 2, 2023 4:56 am

>Studies have shown that only **20%** of a programmer's time is actually spent writing the initial program. The other **80%** is spent on maintenance.

Pareto principle at work.

👍 0

Reply

**Preston West**Reply to [Krishnakumar](#)¹⁹ November 15, 2023 8:19 pm

I wouldn't really attribute the Pareto principle to this. A more fitting example would be if 80% of maintenance costs are attributed to just 20% of a written program, but the 80/20 split here is just on a single factor, a programmer's time. Not really applicable.

👍 0

Reply

**Alex** AuthorReply to [Krishnakumar](#)¹⁹ October 2, 2023 1:47 pm

Yes, indeed. Added as an aside.

👍 0

Reply

**KG BT**Reply to [Alex](#)²⁰ October 24, 2023 4:31 pm

It came to my mind that Pareto principle may be interpreted as a special case of a cognitive bias.

It's not that 80% of time is spent on unimportant stuff — that stuff is needed no less than the 20% part, in order to actually achieve the goal in practice. It's that we, humans, tend to underestimate/dismiss/devalue 80% of "unimportant" job and only see the obvious 20% part. In part, that may explain why we are so bad at predicting time and effort required to complete the job. So, maybe we should answer not x2 but x5 from what we feel, when asked how soon the job will be done =)

👍 1

Reply

**Krishnakumar**Reply to [KG BT](#)²¹ ⌚ November 13, 2023 11:35 am

It was hard to parse your thoughts. I am interested further in this, can you please simplify your explanation a bit more?

👍 0 ➡ Reply

**Abdullah**

⌚ August 27, 2023 12:41 pm

Nice info

👍 0 ➡ Reply

**Ryan**

⌚ August 1, 2023 9:24 am

Alex... I think we know each other... to keep each others privacy I'll just say CCC and 2005. If it doesn't I'll retract the statement.

👍 0 ➡ Reply

**Alex** AuthorReply to [Ryan](#)²² ⌚ August 2, 2023 1:28 pm

Doesn't ring a bell.

👍 0 ➡ Reply

**Eowyn**

⌚ July 28, 2023 7:44 am

Wow, these articles actually make sense! Thank you so much for taking the time (and effort) to create them.

👍 7 ➡ Reply

**Cow**

⌚ July 16, 2023 3:21 pm

It may be worth mentioning that a fixed-width font (in addition to an unambiguous one) is often essential to code formatting. I looked briefly through the other comments to see if anyone had mentioned this, but didn't find anything. Apologies if I missed it.

 Last edited 8 months ago by Cow

 2  Reply



Alex Author

 Reply to Cow²³  July 18, 2023 4:31 pm

Totally! Integrated this into the lesson. Thanks for the feedback.

 1  Reply

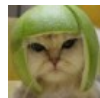


KG BT

 Reply to Alex²⁴  October 24, 2023 4:33 pm

Maybe not at this point, but coding ligatures, like FiraCode font may also worth mentioning.

 0  Reply



Alex Author

 Reply to KG BT²⁵  October 25, 2023 1:50 pm

Added a note about this for advanced readers. Thanks for the suggestion.

 0  Reply



Mehdiali

 July 2, 2023 8:30 pm

Thank you so much for creating this article.

 1  Reply



Kenneth Onuorah

 June 23, 2023 7:00 am

Thanks


 0  Reply



average c enjoyer

 June 9, 2023 7:07 am

I wish you would mention that the term bug has become famous in computer programming due to the use made by Grace Hopper back in the 40s.

 0 Reply**Alex** Author Reply to [average c enjoyer](#)²⁶  June 12, 2023 11:00 am

Is there a particular reason you think this is important to mention? It's not at all clear how much (if any) of a role Hopper had in making the term famous.

 11 Reply**Charity** May 15, 2023 4:56 pm

waw! I really luv this.

 0 Reply**Alexey Suzdalenko** April 16, 2023 2:42 pm

Gracias

 2 Reply**Felipe** March 27, 2023 7:13 am

its soo sad to see have less and less coments in each lesson :(

 7 Reply**Bot** Reply to [Felipe](#)²⁷  May 10, 2023 3:05 am

I guess everyone is too busy reading the chapters.

 2 Reply**brain** Reply to [Bot](#)²⁸  May 23, 2023 1:26 pm

It's true

 0 Reply

**ALEX**

🕒 March 12, 2023 3:16 pm

Спасибо. Очень полезный сайт по обучению C++. Спасибо.



4

➡ Reply

**Kaibao**

🕒 March 10, 2023 2:30 pm

continue studying



2

➡ Reply

**Luis**

🕒 March 2, 2023 12:50 pm

Thanks, it is a good tutorial.



0

➡ Reply

**Alex**

🕒 December 7, 2022 12:42 am

This is the best learning place for c++ as it uses easy to read words and make a meaning out of it



9

➡ Reply

**somia**

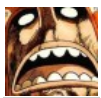
🕒 October 27, 2022 12:57 am

good



2

➡ Reply

**God Usopp**

🕒 October 21, 2022 12:18 pm

Still here. onward ->



7

➡ Reply

**yang**

🕒 September 30, 2022 7:09 am

有加微信一起👉的👉



3



Reply

**Yun**🗨️ Reply to [yang](#)²⁹ 🕒 October 27, 2022 9:13 pm

yes



5



Reply

Links

1. <https://www.learncpp.com/author/Alex/>
2. <https://ikeamenu.com/humix/video/lpms8sWsckf>
3. https://americanhistory.si.edu/collections/search/object/nmah_334663
4. https://en.wikipedia.org/wiki/Pareto_principle
5. <https://www.codingfont.com/>
6. <https://github.com/tonsky/FiraCode>
7. <https://www.learncpp.com/cpp-tutorial/introduction-to-the-compiler-linker-and-libraries/>
8. <https://www.learncpp.com/>
9. <https://www.learncpp.com/cpp-tutorial/introduction-to-cplusplus/>
10. <https://www.learncpp.com/introduction-to-cpp-development/>
11. <https://www.learncpp.com/cpp-tutorial/installing-an-integrated-development-environment-ide/>
12. <https://gravatar.com/>
13. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-593729>
14. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-593148>
15. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-592545>
16. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-591120>
17. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-589212>
18. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-588960>
19. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-588044>
20. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-588123>
21. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-589078>
22. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-584883>
23. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-583904>
24. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-584018>
25. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-589079>

26. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-581494>
27. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-578720>
28. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-580241>
29. <https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/#comment-573670>